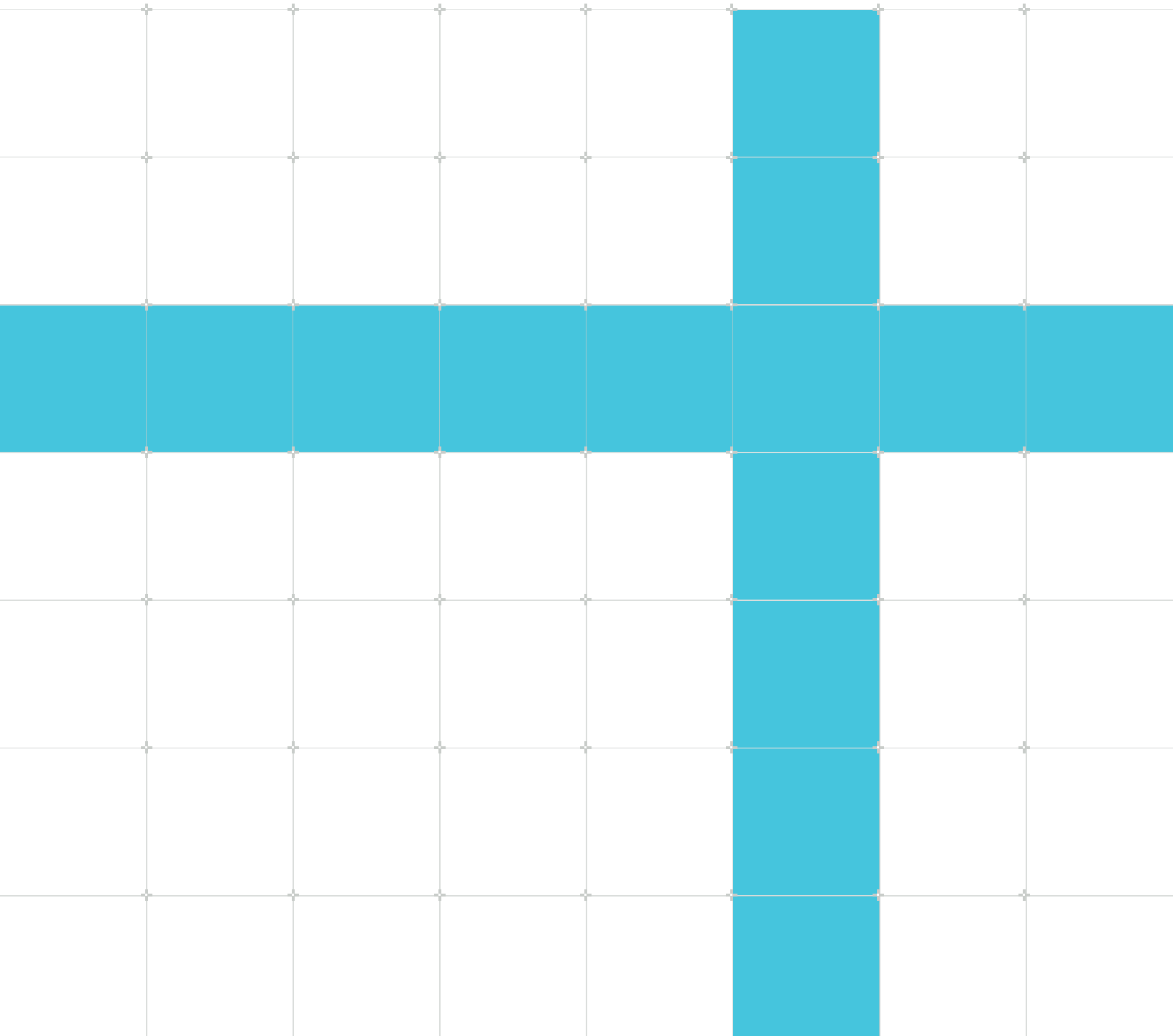
arm

Introducing the R-Profile

1.0

architecture guide

Issue
DEN0130_0100_en



Introducing the R-Profile
architecture guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100	24 February 2022	Non-Confidential	Initial release

Proprietary Notice

and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

This guide introduces you to the R-Profile of the Arm architecture. You will learn about the Arm architecture, the evolution of the R-Profile, and the features in all R-Profile versions.

This guide does not provide detail about individual features or give programming examples. The guide is intended as a high-level introduction comparing the different version of the architecture, with links to other resources to learn more.

This guide is aimed at users working with systems based on R-Profile processors. No prior knowledge of the Arm architecture is needed, but a general familiarity with processors, programming, and programming terminology is assumed.

2. About the Arm architecture

The Arm architecture is one of the most popular processor architectures in the world today, with several billion Arm-based devices shipped every year.

The following table describes the Arm architecture profiles and how the profiles are used:

A-Profile (Applications)	R-Profile (Real-time)	M-Profile (Microcontroller)
High performance	Targeted at systems with real-time requirements	Smallest and lowest power. Used in small, highly power-efficient devices.
Designed to run complex operating systems such as Linux or Windows	Commonly found in networking equipment and embedded control systems	Found in many Internet of Things (IoT) devices

These different profiles share several base features, even though they tailor the Arm architecture to the needs of different use cases.

This guide describes the R-Profile of the Arm architecture. The R-Profile targets use cases where real-time performance is important, with a feature set optimized for deterministic timing and better interrupt latency. R-Profile processors are widely used in many different markets, like networking and automotive.

What is architecture?

Architecture is a functional specification. The Arm architecture is a functional specification for a processor. The architecture specifies how a processor behaves, including the available instructions, and what those instructions do.

You can think of an architecture as a contract between the hardware and the software. The architecture describes the functionality the software can rely on the hardware to provide.

The architecture specifies the following functionality:

- Instruction set
  - The function of each instruction
  - How that instruction is encoded, or represented in memory
- Register set
  - How many registers there are
  - The size of the registers
  - The function of the registers
  - Their initial state
- Exception model
  - The different levels of privilege
  - The types of exceptions
  - What happens when taking or returning from an exception

- Memory model
  - How memory accesses are ordered
  - How the caches behave, when and how software performs explicit maintenance
- Debug, trace, and profiling
  - How breakpoints are set and triggered
  - What information is captured by trace tools and in what format

Architecture and micro-architecture

The architecture covers the functional behavior of a processor. The design of a processor is described as micro-architecture.

Micro-architecture includes the following:

- Pipeline length and layout
- Number and sizes of caches
- Cycle counts for individual instructions
- Which optional features are implemented

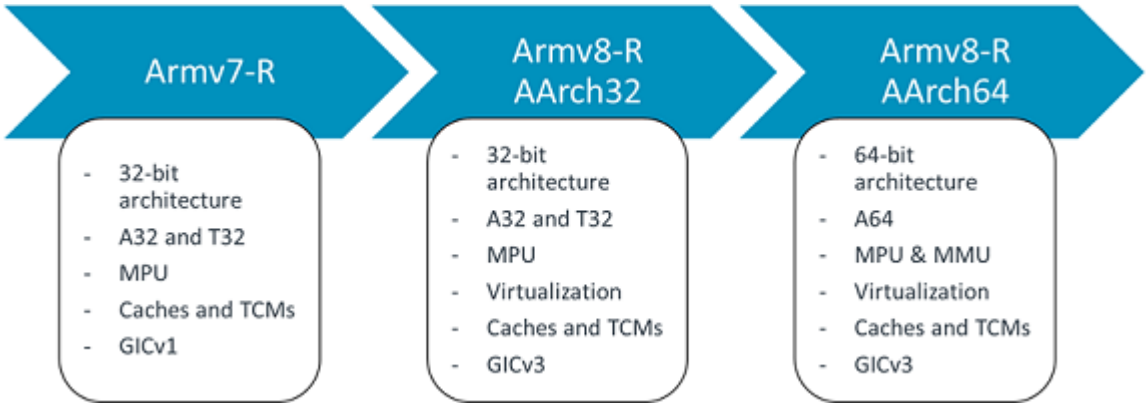
For example, the Cortex-R4 and Cortex-R8 are both implementations of the Armv7-R architecture. This means that these processors have the same the architecture, but they have different micro-architectures. The following table gives examples of some of the micro-architectural differences:

Micro-architecture	Cortex-R4	Cortex-R8
Pipeline	In-order, 8 stages	Out-of-order, variable length
Number of cores	1	Configurable, 1-4
Caches	L1 instruction cache: 0-64KiB, L1 data cache: 0-64KiB	L1 instruction cache: 0-64KiB, L1 data cache: 0-64KiB
Tightly Coupled Memories	A TCM: 0-8MB, B TCM: 0-8MB	Instruction TCM: 0-1MB, Data TCM: 0-1MB

R-Profile features

The R-Profile was introduced in Armv7. The following diagram shows how the architecture evolved from Armv7 to Armv8:

Figure 2-1: R-Profile architecture



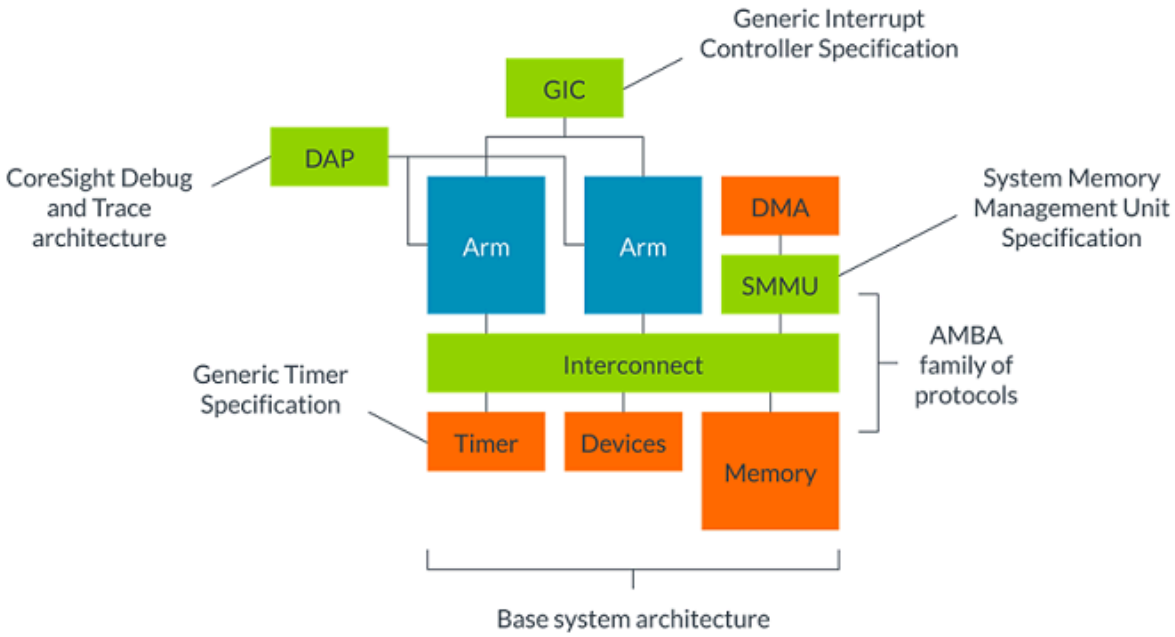
The following is a summary of the features in each version:

- Armv7-R
  - 32-bit architecture
  - A32 and T32 instruction sets, including optional Floating-Point. A32 and T32 were previously known as the Arm and Thumb instruction sets.
  - Optional Memory Protection Unit (MPU). When no MPU is present, a fixed set of attributes is used.
  - Support for GICv1
- Armv8-R AArch32
  - 32-bit architecture
  - Virtualization support
  - A32 and T32 instruction sets, including optional Floating-Point and Neon
  - Stage 1 and stage 2 MPUs
  - Support for GICv3
- Armv8-R AArch64
  - 64-bit architecture
  - Virtualization support
  - A64 instruction set, including optional Floating-Point, Neon, and SVE extensions
  - Stage 2 MPU, with stage 1 MPU or Memory Management Unit (MMU)
  - Support for GICv3

R-Profile architectures

A system includes more than just a processor. As well as processor architectures, Arm produces architectures for other common system components. The following simplified system diagram highlights components and architectures that are relevant to the R-Profile:

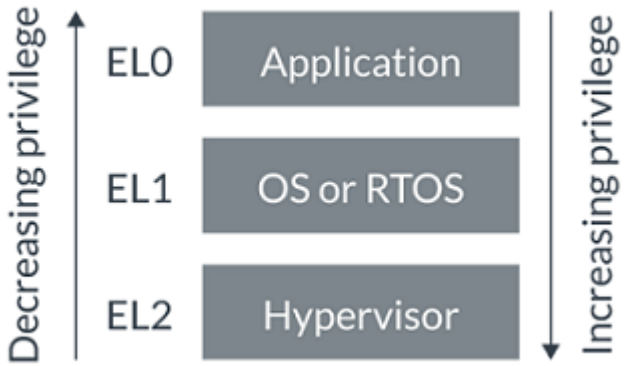
Figure 2-2: Other Arm specifications in an SoC



3. Armv8-R AArch64

Armv8-R AArch64 is the latest version of the R-Profile and inherits the 64-bit AArch64 execution environment from Armv8-A. Armv8-R AArch64 has three privilege levels known as Exception levels, as shown in the following diagram:

Figure 3-1: Exception levels



Unlike the A-Profile, the R-Profile only supports a single Security state. The single Security state in R-Profile is equivalent to the Secure state in A-Profile. For example, both the Secure and Non-secure physical address spaces are supported.

Armv8-R AArch64 supports the A64 instruction set, which is a fixed-length 32-bit instruction set. Armv8-R AArch64 does not support A32 (Arm) or T32 (Thumb).

This guide does not cover the details of the AArch64 Exception model and A64 instruction set. Information on these topics can be found in the following guides:

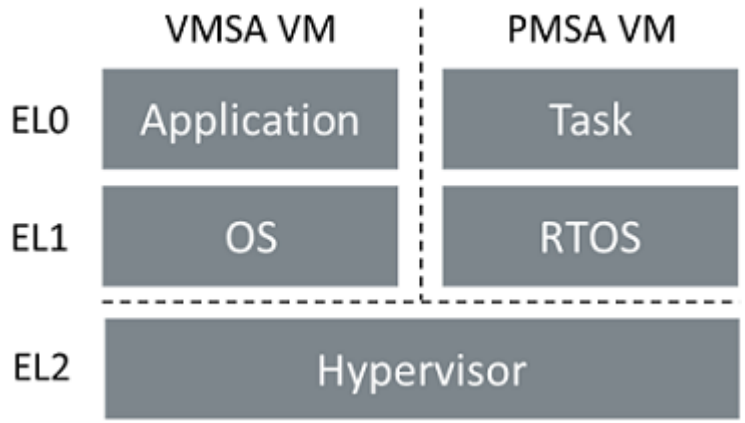
- Learn the architecture: AArch64 Exception model
- Learn the architecture: AArch64 Instruction Set Architecture
- Neon Programmer's Guide for Armv8-A

Virtualization

Virtualization allows multiple guest virtual machines (VMs) to be hosted on the same physical machine. EL2 provides controls for virtualization features in the processor. Software running at EL2, referred to as a hypervisor, is responsible for creating and scheduling VMs. The VM runs at EL1 and EL0 and can contain bare-metal code, an RTOS, or a rich OS such as Linux.

Armv8-R AArch64 supports virtualization, as shown in the following diagram:

Figure 3-2: Virtualization in Armv8-R AArch64



The virtualization support in the processor includes the following:

- Traps on EL1 access to control registers. This allows the hypervisor to emulate some operations with a VM.
- Stage 2 MPU. This allows the hypervisor to control which system resources are visible to a VM. For example, preventing one VM from accessing resources allocated to a different VM.
- Virtual interrupt support. The hypervisor can control which interrupts are presented to each VM.

More information on traps, virtual interrupts, and virtual timers can be found in the following guides:

- Learn the architecture: AArch64 Virtualization
- Learn the architecture: Arm Generic Interrupt Controller

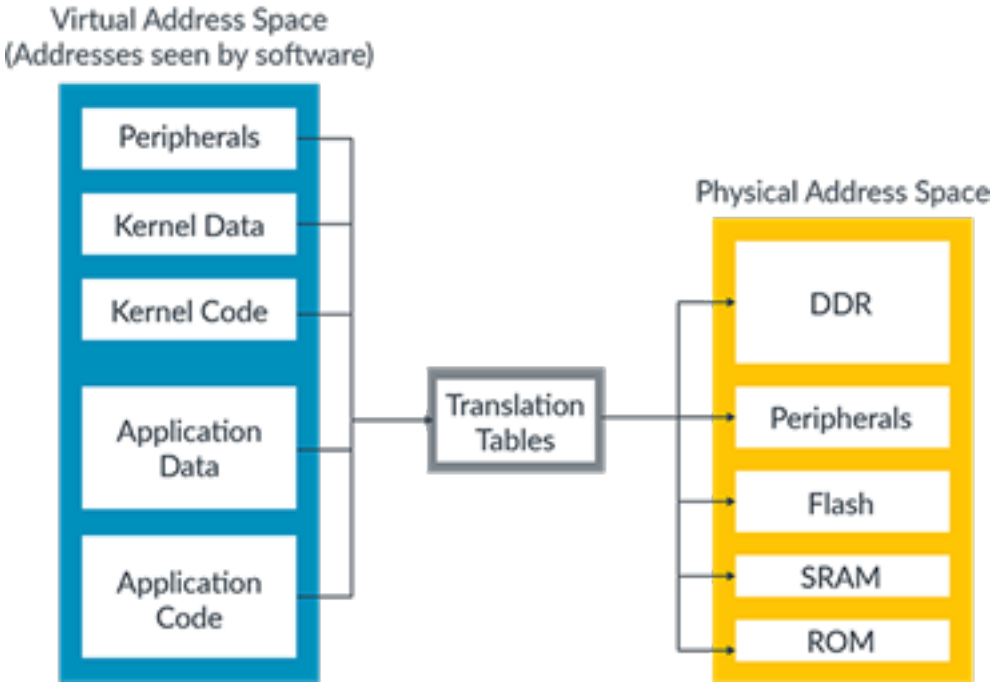
MPU and MMU

Traditionally, the R-Profile of the Arm architecture only supported Memory Protection Units (MPU) for memory management. An MPU uses registers to define regions with associated properties such as read/write permissions and cache policies. When the processor accesses a memory location, the MPU checks the access against the properties for the region the address falls within. This model is known as the Physical Memory System Architecture (PMSA).

Armv8-R AArch64 adds support for the Virtual Memory System Architecture (VMSA). VMSA is also used in the A-profile and replaces the MPU with a Memory Management Unit (MMU).

With VMSA, the addresses seen by software are referred to as virtual addresses. When software accesses memory, the issued address is processed by the MMU. The MMU uses a set of software-controlled tables (translation tables) to translate the virtual address seen by software into a physical address. This physical address is understood by the SoC's memory system and the process is shown in the following diagram:

Figure 3-3: VMSA process



The benefit of using virtual addresses is that it allows management software to control the view of memory that is presented to software. The management software can control what memory is visible, the virtual address at which that memory is visible, and what accesses are permitted to that memory. For example, an operating system can sandbox applications (hide the resources of one application from another application) to provide abstraction from the underlying hardware.

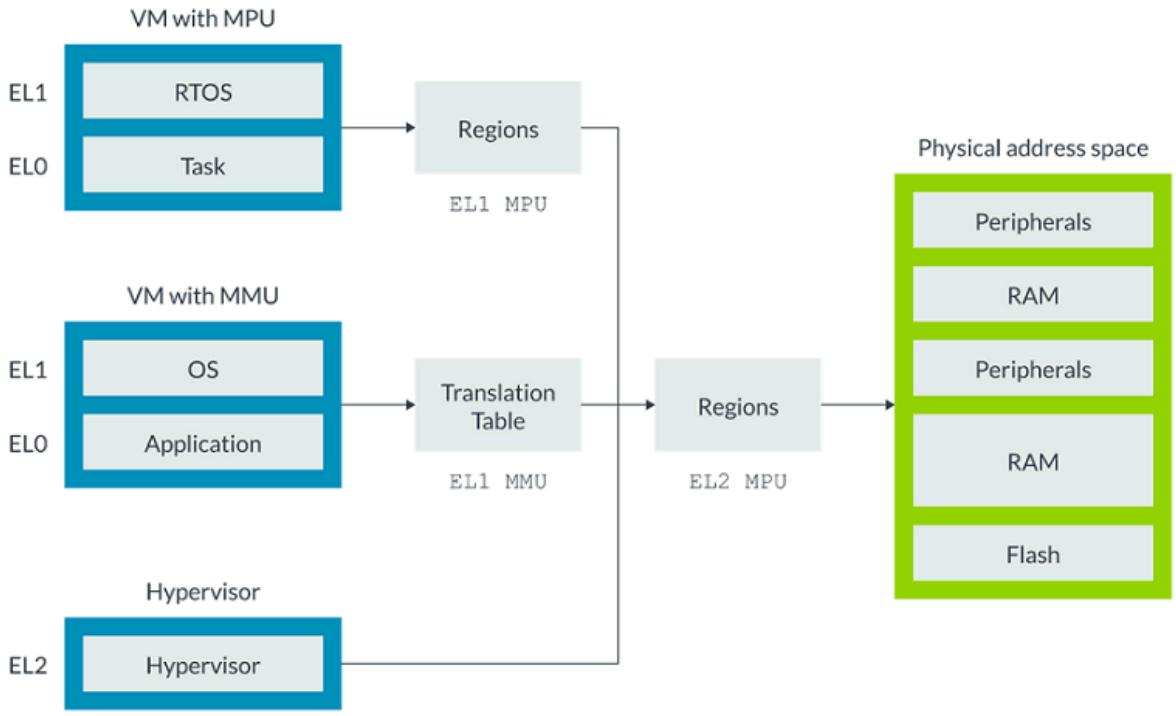
The MMU provided by VMSA is more flexible than an MPU, supporting address translation as well as applying attributes and permissions. However, the MMU relies on translation tables held in memory, with recently used translations cached by the processor. As a result, the time taken to check a memory access depends on whether the translation is cached and if not, the time required to walk the translation tables.

The MPU provided by PMSA is more limited, supporting memory partitioning but not translation. However, all information needed to perform an MPU check is held in the processor's registers, providing consistent and deterministic timing.

Typically, a rich operating system such as Linux requires an MMU to function. For RTOSs, an MPU is sufficient and preferred.

Armv8-R AArch64 supports a Stage 2 MPU with an MMU or MPU at Stage 1, as shown in the following diagram:

Figure 3-4: Two stage memory protection



The Stage 1 MPU or MMU is configured by the RTOS or OS within a VM, and controls accesses to resources visible to that VM.

The Stage 2 MPU is configured by the hypervisor, controlling the resources visible to each VM. The Stage 2 MPU is also used to check accesses by the hypervisor.

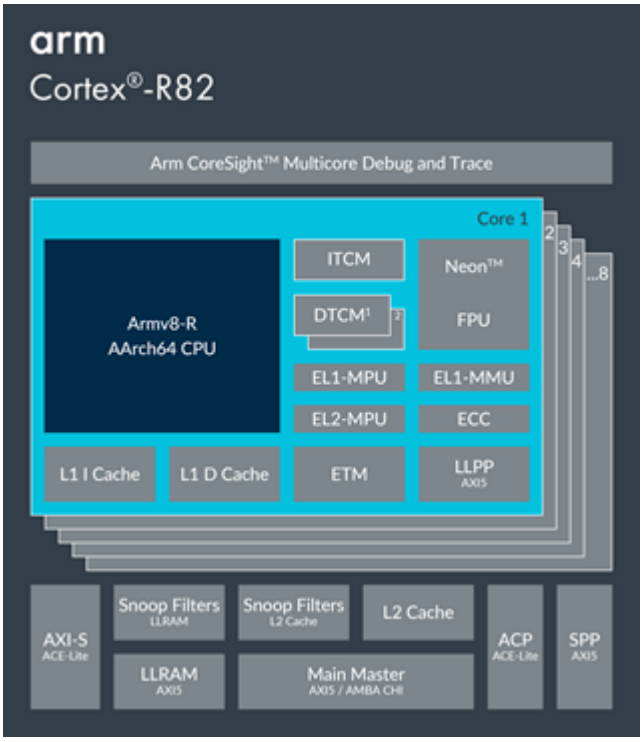
Armv8-R AArch64 support for PMSA and VMSA gives flexibility to software. AArch64 allows a combination of OS and RTOS to run in different VMs, with the hypervisor managing access to system resources.

For more information about AArch64, see AArch64 memory management.

Cortex-R82

The Cortex-R82 is the first processor to implement Armv8-R AArch64. It is an in-order, superscalar, cluster processor that can address up to 1TB of memory. The following diagram shows the Cortex-R82 architecture:

Figure 3-5: Cortex-R82 architecture



For more information, see the Cortex-R82 product page.

4. Armv8-R AArch32

Armv8-R AArch32, originally referred to as Armv8-R, was first announced in 2013. Building on Armv7-R, Armv8-R AArch32 added support for virtualization. Processors based on Armv8-R AArch32 are widely deployed across the Arm ecosystem.

Programmer's model

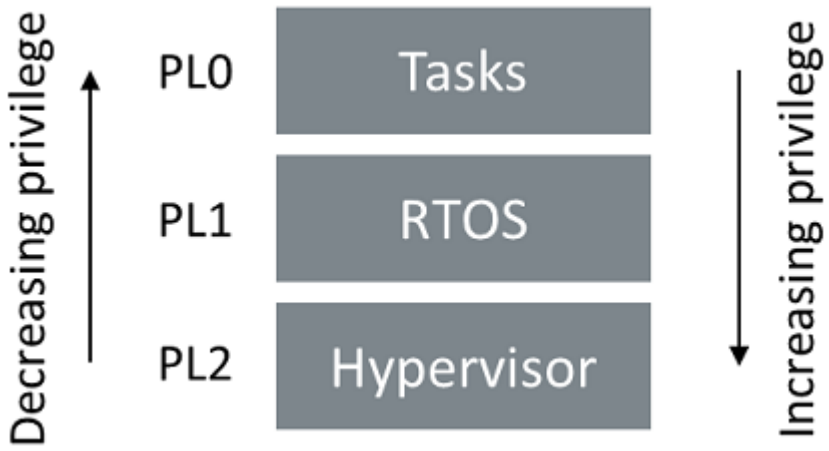
Armv8-R AArch32 uses the AArch32 Execution state also found in Armv8-A, which gives backwards compatibility with Armv7-A/R.

Armv8-R supports the following levels of privilege:

- PL0 is the least privileged and where applications or tasks typically execute
- PL1 is where management software such as an RTOS operates
- PL2 is the most privileged and where the software that controls the processor's virtualization features executes. This software is typically referred to as a hypervisor.

The privilege levels are illustrated in the diagram:

Figure 4-1: A diagram showing privilege levels in Armv8-R AArch32



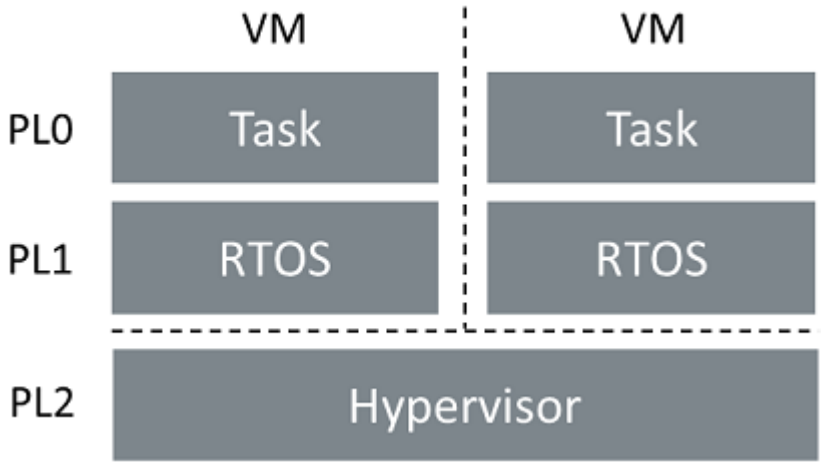
Like Armv8-R AArch64, Armv8-R AArch32 supports a single Security state.

Virtualization

Virtualization allows multiple guest VMs to be hosted on the same physical machine. EL2 provides controls for virtualization features in the processor. Software running at PL2 or the hypervisor is responsible for creating and scheduling VMs.

The VM runs at PL1 and PL0 and can contain bare-metal code and RTOSs, as shown in the following diagram:

Figure 4-2: A diagram showing virtualization architecture



The virtualization support in the processor includes the following:

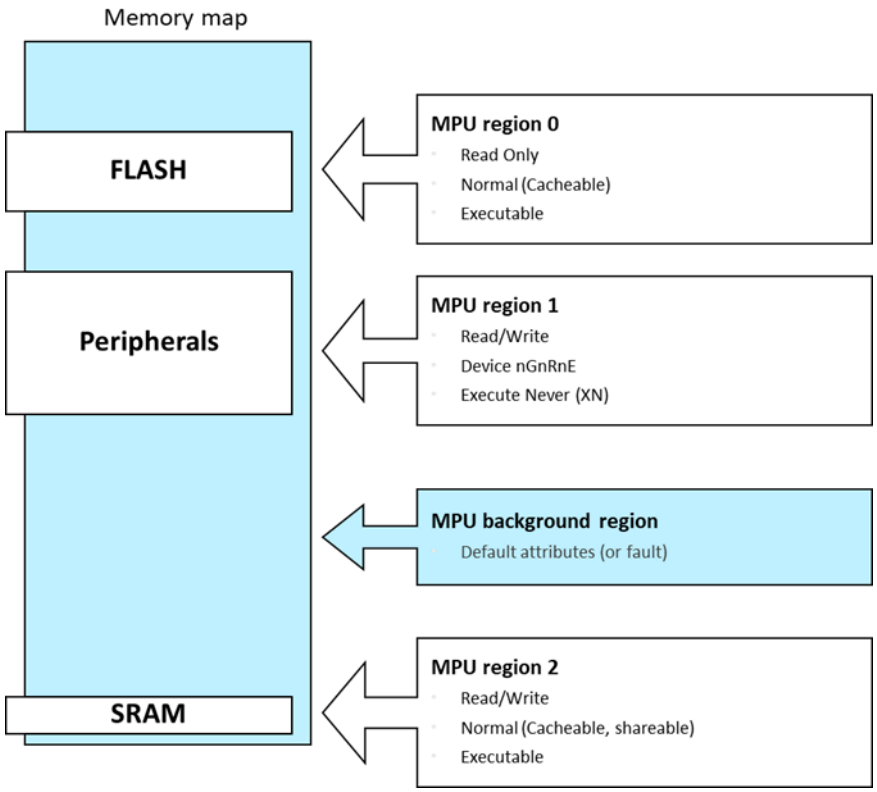
- Traps on PL1 accesses to control registers. This allows the hypervisor to emulate some operations with a VM.
- Stage 2 MPU. This allows the hypervisor to control which system resources are visible to a VM. For example, preventing one VM from accessing resources allocated to a different VM.
- Virtual interrupt support. The hypervisor can control which interrupts are presented to each VM.

MPU

Armv8-R AArch32 implements the Physical Memory System Architecture (PMSA), providing an MPU to control memory protection.

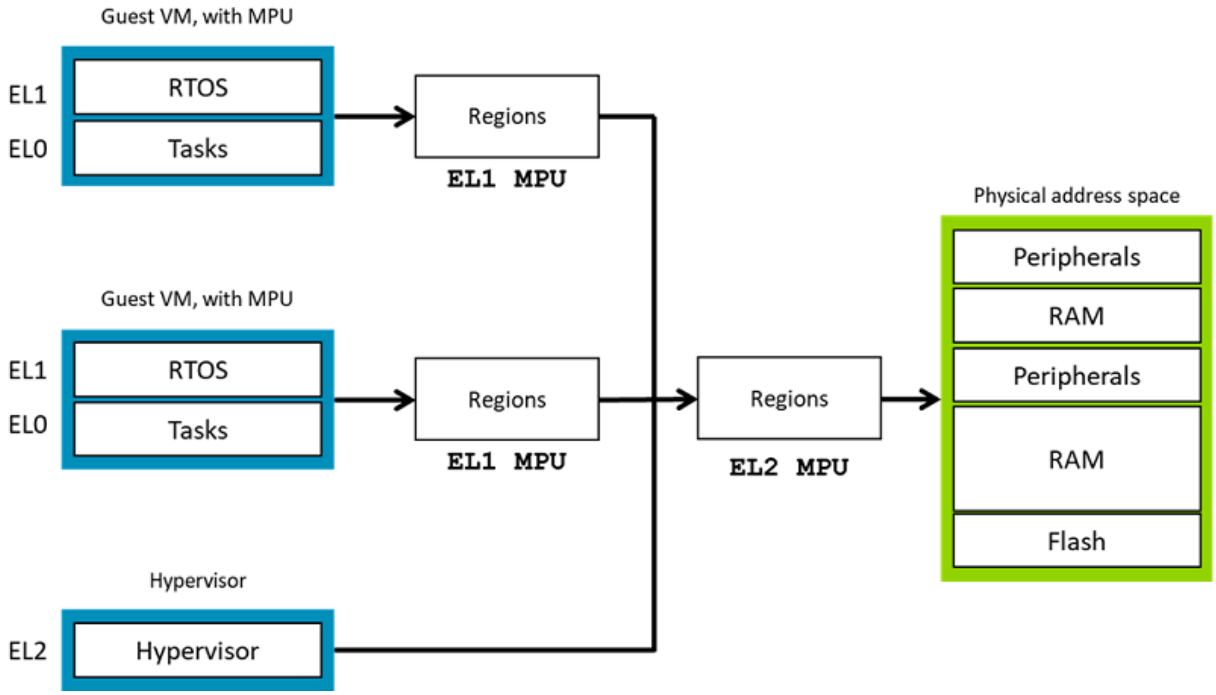
An MPU uses registers to define regions with associated properties such as read/write permissions and cache policies. When the processor accesses a memory location, the MPU checks the access against the properties for the region the address is in. This model is known as the PMSA. The following diagram shows a simple set of regions that can be defined in the MPU:

Figure 4-3: A diagram showing the regions defined in the MPU



Armv8-R AArch32 implements the PMSA, providing Stage 1 and Stage 2 MPUs. This memory protection is shown in the following diagram:

Figure 4-4: A diagram showing two stage memory protection in Armv8-R AArch32



The Stage 1 MPU is configured by the RTOS in a VM and controls access to the resources visible to that VM.

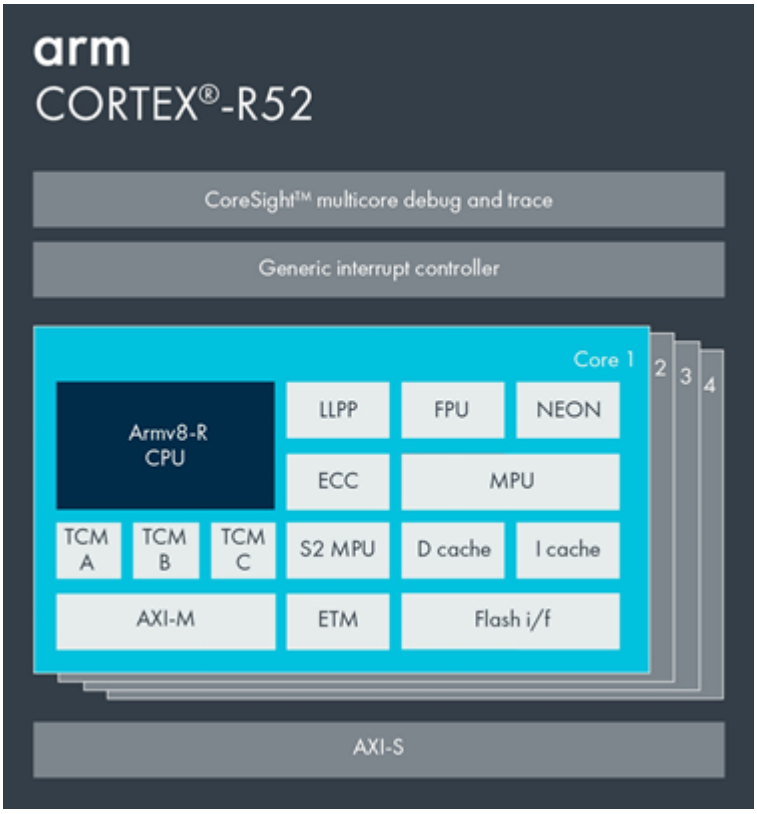
The Stage 2 MPU is configured by the hypervisor, controlling the resources are visible to each VM. The Stage 2 MPU is also used to check accesses by the hypervisor itself.

Unlike Armv8-R AArch64, there is no support for VMSA or MMUs in Armv8-R AArch32.

Cortex-R52 and Cortex-R52+

The Arm Cortex-R52 processor was the first Armv8-R AArch32 implementation. Cortex-R52 delivers high-performance 32-bit processing with efficient code density, combined with the highest level of integrated capability for functional safety of any Arm processor.

Figure 4-5: A diagram showing the Cortex-R52 processor



The Arm Cortex-R52+ processor is software compatible with Cortex-R52, providing improved configurability for real-time applications with functional safety requirements such as separation of VMs.

Both Cortex-R52 and Cortex-R52+ include integrated Generic Interrupt Controllers (GIC), which implement the GICv3 architecture.

For more information, see the following product pages:

- Cortex-R52+
- Cortex-R52

5. Armv7-R

Armv7-R was first introduced in 2006 with the release of the Cortex-R4.

Programmer's model

Armv7-R implements a 32-bit architecture, compatible with AArch32.

Armv7-R supports two levels of privilege, PL1 and PL2, as shown in the diagram:

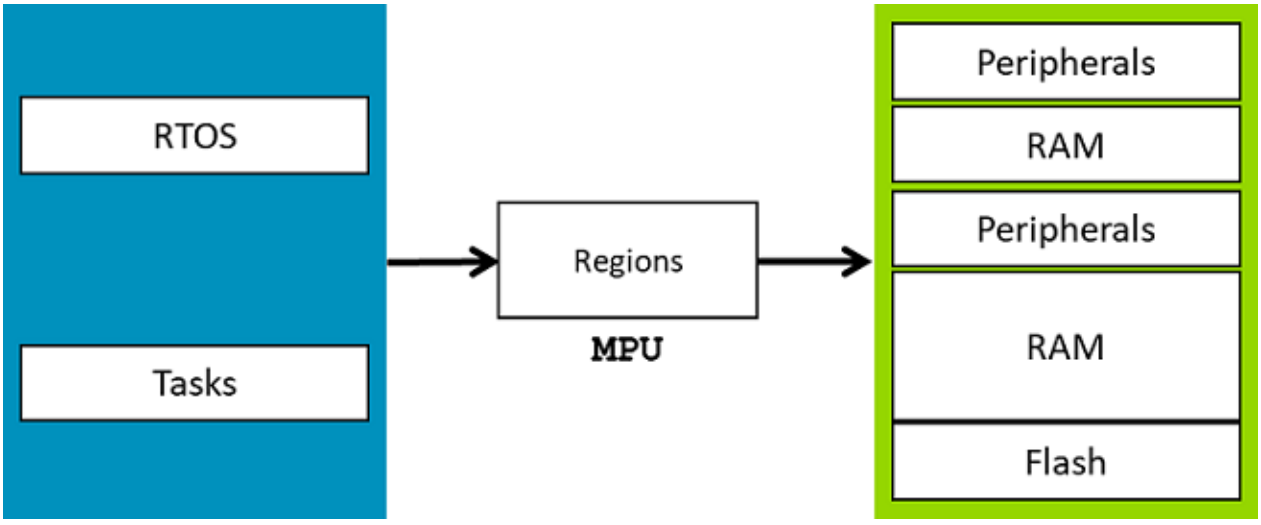
Figure 5-1: A diagram showing Armv7-R privilege

For a detailed introduction to the Armv7-R programmer's model, see the Arm Cortex-R Series Programmer's Guide.

MPU

Armv7-R implements the PMSA with a single Stage 1 MPU, as shown in the following diagram:

Figure 5-2: A diagram showing the PMSA architecture



For more information about processors that implement the Armv7-R architecture, see the following product pages:

- Cortex-R4
- Cortex-R5
- Cortex-R7
- Cortex-R8

6. Documentation

This section describes the available documentation relating to Cortex-R processors and the information contained in each document:

- Arm Architecture Reference Manual Armv8, for A-profile architecture
- Arm Architecture Reference Manual Supplement - Armv8, for Armv8-R AArch64 architecture profile
- Arm Generic Interrupt Controller v3 and v4 Version 3.2
- Arm Cortex-R52 Processor Technical Reference Manual
- Configuration and Integration Manual
- SoC datasheet

The Cortex-R processor Configuration and Integration Manual (CIM) describes how to integrate the processor into a system. Generally, this information is only relevant to SoC designers. The CIM is only available to licensees and is delivered as part of the RTL delivery bundle which licensees download. Licensees can find it in the `docs` or `documentation` subdirectory of the extracted RTL bundle. The exact location of the IIM in the RTL delivery bundle depends on the Cortex processor that you are using.

If you are working with an existing SoC, you also use documentation from the SoC's manufacturer. This documentation is typically referred to as a datasheet. The datasheet gives information specific to that SoC.

The following table gives examples of the information found in each type of guide:

Information type	Arm Architecture Reference Manual	Armv8-R AArch32 supplement	Arm Generic Interrupt Controller specification	Technical Reference Manual	Configuration Integration Manual	Datasheet
Instruction set	X					
Instruction cycle timings				X		
Architectural registers	X	X	X (for GIC registers)			
Processor-specific registers				X		
MPU		X				
Exception model	X					
Support for optional features				X	X (synthesis choice)	
Power management				X		
Bus ports and signals				X	X	
Bus transactions generated by processor				X		
Memory map						X

Information type	Arm Architecture Reference Manual	Armv8-R AArch32 supplement	Arm Generic Interrupt Controller specification	Technical Reference Manual	Configuration Integration Manual	Datasheet
Peripherals						X
Pin-out of SoC						X

7. Related information

Here are some resources related to material in this guide:

- For Armv7-R, see the ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition and ARMv7-R - Hidden Power
- For Armv8-R, see the following guides:
  - Arm Architecture Reference Manual Armv8, for A-profile architecture
  - ARM Architecture Reference Manual Supplement - ARMv8, for the ARMv8-R AArch32 architecture profile
  - Arm Architecture Reference Manual Supplement - Armv8, for Armv8-R AArch64 architecture profile
- Cortex-R82 product page
- For information on traps, virtual interrupts, and virtual timers see AArch64 Virtualization and Arm Generic Interrupt Controller v3 and v4 Version 3.2

8. Next steps

This guide introduced you to the R-Profile of the Arm architecture. You learned about the Arm architecture, the evolution of the R-Profile, and the features in all R-Profile versions.

As a next step, you can learn more about the Cortex-R processor by reading the guides listed in Documentation. To compare the features in all Cortex processors, see the Arm Cortex-R Processor Comparison Table.